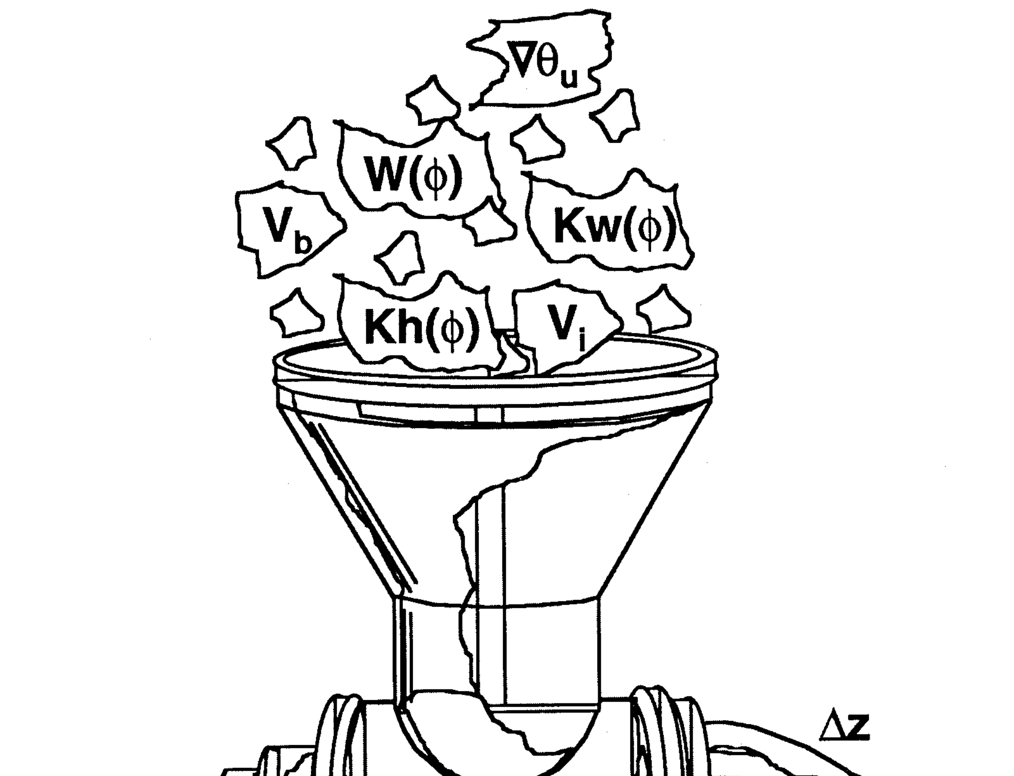
CRREL REPORT 95-12

CRREL

RIGIDICE Model of Secondary Frost Heave

Patrick B. Black

May 1995



Abstract
A revised version of an earlier attempt to numerically solve Miller's equations for the RIGIDICE model of frost heave is presented that corrects earlier mistakes and incorporates recent improvements in the scaling factors of ground freezing. The new version of the computer code also follows the concepts of Object Oriented Numerics (OON), which allow for easy modification and enhancements. Analysis of the program is accomplished with the symbolic math program MathCad. A brief sensitivity analysis of the input variables indicates that those parameters that calculate the hydraulic conductivity have the greatest influence on the variability of predicted heaving pressure.

For conversion of SI metric units to U.S./British customary units of measurement consult ASTM Standard E380-89a, *Standard Practice for Use of the International System of Units,* published by the American Society for Testing and Materials, 1916 Race St., Philadelphia, Pa. 19103.

CRREL Report 95-12

US Army Corps of Engineers
Cold Regions Research & Engineering Laboratory

RIGIDICE Model of Secondary Frost Heave

Patrick B. Black

May 1995

Prepared for
OFFICE OF THE CHIEF OF ENGINEERS

Approved for public release; distribution is unlimited.

PREFACE

This report was prepared by Dr. Patrick B. Black, Soil Physicist, Applied Research Branch, Experimental Engineering Division, U.S. Army Cold Regions Research and Engineering Laboratory. Funding was provided by U.S. Army Project 4A762784AT42, *Pavements in Cold Regions* (231), Task BS, Work Unit AT42-CP-C01, *Freezing and Thawing of Soil Water Systems*.

Technical review was provided by Dr. S. Colbeck and Dr. R. Berg, both of CRREL.

The contents of this report are not to be used for advertising or promotional purposes. Citation of brand names does not constitute an official endorsement or approval of the use of such commercial products.

CONTENTS

ILLUSTRATIONS

Figure

TABLES

Table

NOMENCLATURE

Symbol	*Units*	*Definition*
G, W, I	$m^3\,m^{-3}$	volume fractions of grains, water and ice
W_n, W_{n+1}	$m^3\,m^{-3}$	volumetric water content within layers n and n+1 in the fringe
e		porosity
W_{sat}, W_d	$m^3\,m^{-3}$	volumetric water contents at saturation and at the lower limit of freezing
W_f, I_f	$m^3\,m^{-3}$	volumetric water content and ice content of the frozen soil existing between two mature ice lenses
v_i	$m\,s^{-1}$	heave rate
v_b	$m\,s^{-1}$	penetration rate of freezing front
q_w	$m^3\,m^{-2}\,s^{-1}$	flux of water
$(q_w)_b$	$m^3\,m^{-2}\,s^{-1}$	flux of water into the fringe
$(q_w)_n, (q_w)_{n+1}$	$m^3\,m^{-2}\,s^{-1}$	flux of water into a layer in the fringe
k_w	$m\,s^{-1}$	hydraulic conductivity
$(k_w)_{sat}$	$m\,s^{-1}$	saturated hydraulic conductivity
α, β	—	Brooks and Corey coefficients
u_w, u_i	Pa	water and ice gauge pressures
ϕ	Pa	capillary pressure (u_i–u_w)
ϕ_b	Pa	capillary pressure at base of fringe (i.e., ice entry pressure)
ϕ_n, ϕ_j	Pa	capillary pressures within layers n and j in the fringe
$\sigma_T, \sigma_e, \sigma_n$	Pa	total, effective and neutral stresses as expressed by the Terzaghi equation
χ	—	Snyder-Bishop stress partition factor
f_w	$N\,m^{-3}$	body force on liquid water
q_h	$W\,m^{-2}$	flux of heat
$(q_h)_n, (q_h)_{n+1}$	$W\,m^{-2}$	flux of heat within layers n and n+1 in the fringe
$(q_h)_b, (q_h)_f$	$W\,m^{-2}$	flux of heat through bottom of fringe and upper boundary zone
$(k_h)_g, (k_h)_w, (k_h)_i$	$W\,K^{-1}\,m^{-1}$	thermal conductivities of grains, water and ice
θ	°C	temperature
θ_0	K	absolute temperature of a flat ice/water interface in equilibrium at standard conditions
h	$J\,m^{-3}$	volumetric latent heat of fusion
Y	—	specific gravity of ice
z	m	position positive up from the bottom of the fringe
$(d\theta/dz)_b, (d\theta/dz)_f$	$°C\,m^{-1}$	temperature gradient at base of fringe and in upper boundary zone

RIGIDICE Model of Secondary Frost Heave

PATRICK B. BLACK

INTRODUCTION

In an earlier paper, Black and Miller (1985) reported on progress to numerically solve the differential equations of secondary frost heave (Miller 1978) using a simplified approach. They presented a series of scenarios that predicted anticipated results. The program was experimentally confirmed by comparing observed behavior of independently conducted frost heave tests to model predictions using hydraulic and thermal properties for the test soil. They found that the model indicated that the experimental data were flawed in their stress measurements, but agreed within acceptable tolerance with the thermal measurements. No additional effort was made at that time to extend the analysis or increase the utility of the program.

This report is an extension of the earlier work. It presents improvements in two areas. First, a minor mistake in the calculation of the water flux is corrected; also, the current formulations for the scaled equations of ground freezing are used (Miller 1990). Second, the computer code is presented in a form readily available for enhancements. This is accomplished by employing the concepts of Object Oriented Numerics (OON), which allow the code to be easily added to without directly changing the original source code (Wong et al. 1993).

The concepts and equations of the RIGIDICE (Black and Miller 1985) model are first reviewed. The new C++ formulation of the code is then discussed and presented in its entirety in Appendix A. The code is linked as a Dynamically Linked Library (DLL) that is attached to MathCad 5.0+ (MathSoft 1994). A preliminary sensitivity study is conducted to examine the influence of uncertainty in input parameters on the variability of the calculated output parameters.

RIGIDICE

Black and Miller (1985) simplified the solution of the differential equations for secondary frost heave (Miller 1978) by employing two physical assumptions and one numerical trick. Their model, as well as this model, is valid for one-dimensional, incompressible, air-free and solute-free soil. The solute-free restriction assumes that no additional osmotic gradient is imposed over the ever present osmotic gradient in the double layer of the unfrozen water surrounding the grains. The incompressible restriction excludes the process of consolidation at this time. Finally, the air-free condition states that the region undergoing heave must be water saturated. This makes perfect sense, since air does not freeze, which means that the soil must have been saturated, or within 10%, because of the volume expansion necessary for a lens to appear. These restrictions are minor since they result in the model predicting the behavior of the highly frost-susceptible fine-grain silts.

First physical approximation (lensing cycle)

First, the progression of frost heaving through soil is approximated physically by a series of independent lensing cycles. A lensing cycle is defined to be the time step that begins with the formation of a lens and ceases with the initiation of a new lens. During each individual lensing cycle, the frozen soil above the fringe is composed of a series of identical layers of ice and frozen soil as depicted in Figure 1. The thickness of these lenses and the frozen soil between them is equal to the thickness of the mature lens and interspaced frozen soil at the end of the lensing cycle.

This approximation bounds the fringe between the lower freezing front and the upper boundary

zone. The freezing front is the plane that separates soil containing ice from the unfrozen soil. The upper boundary zone is not defined by a plane, but is rather a zone that always contains a thickness of pure ice equal to a mature ice lens and a thickness of frozen soil equal to the thickness of the mature interspace frozen soil. Figure 1 also shows the location of these regions at various stages throughout the lensing cycle.

Under steady-state conditions, the inputs across the freezing front are given by

grains	Gv_b
water	$W_{sat}v_b + (q_w)_b$
ice	0
sensible heat	$(q_h)_b$
latent heat	$h\,[W_{sat}v_b + (q_w)_h]$

and the outputs across the boundary zone are

grains	Gv_b
water	$W_d v_b$
ice	$I_f v_b + v_i$
sensible heat	$(q_h)_f$
latent heat	$h\,[W_f v_h]$.

The global mass balance is therefore

$$(q_w)_b = Yv_i + \left[(Y-1)(W_{sat} - W_f) \right] v_b \qquad (1)$$

and heat balance is

$$(q_h)_f = (q_h)_b + h\left[(W_{sat} - W_f) v_b + (q_w)_b \right]. \qquad (2)$$

In its purest sense, this results in six unknowns and two equations. In practice, though, several of the variables are known from other relationships or observations. The numerical trick is to specify at least three of the variables and guess another to start the numerical solution process.

Temperature profiles during heaving are the most common piece of experimental information collected because it is easy to measure. Freezing front penetration rate is readily determined from these data, so v_b is one logical variable to specify. Heat fluxes are also readily determined from temperature profile data if the thermal conductivities are known. The thermal conductivity of the unfrozen soil is a constant in this one-dimensional case because of the air-free restriction. The sensible heat flux across the frozen fringe $(q_h)_b$ therefore makes another reasonable choice. Unfortunately, there is no a priori justification for choosing from the remaining variables. The thermal conductivity in the frozen soil could be computed if the lens thickness and spacing and the residual unfrozen water content W_f were known. The flux of water into the fringe could also be computed if the amount of heave were known. For lack of any further justification, the rate of heave v_b *is* fixed.

If the residual unfrozen water W_f in the frozen soil between ice lenses is known, then the remain-

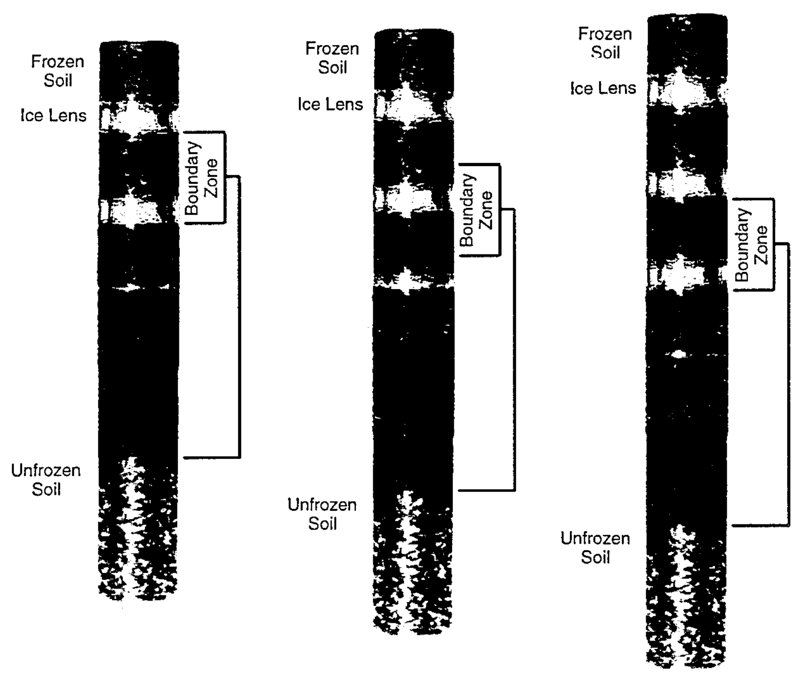


Figure 1. Lensing cycle.

ing variables are known for a given soil. As a first approximation, the amount of residual unfrozen water is assumed to be equal to the lower limit of freezing for the soil. In some situations, this might represent all the required information for the modeler. More often, the heaving pressure, and lens thickness and spacing are also required.

Second physical approximation (equivalence of instantaneous and averaged fluxes)

To calculate the heaving pressure and lens location within the fringe, profiles of temperature and pressures must be calculated throughout the fringe. To perform such calculations, a method of executing mass and energy balances within the fringe must be obtained. This is accomplished with the second physical approximation. It states that the instantaneous fluxes of matter and energy at the beginning and the end of the lensing cycle are equal to the averaged values of the fluxes during the lensing cycle.

Another way of stating the second approximation is that any instantaneous fluctuations in magnitude of the mass and energy fluxes during the lensing cycle are negligible. This means that the magnitude of the penetration rate, heave rate and temperature gradient within the unfrozen soil are invariant with time and space. In finite difference form, the local mass balance within the fringe is

$$(q_{\mathrm{w}})_{\mathrm{n}+1} = (q_{\mathrm{w}})_{\mathrm{n}} + v_b - Y(v_b + v_i)[W_{\mathrm{n}} - W_{n+1}] \qquad (3)$$

and thermal balance is

$$(q_{\mathrm{h}})_{\mathrm{n}+1} = (q_{\mathrm{h}})_{\mathrm{n}} + h\left[(W_{\mathrm{n}} - W_{\mathrm{n}+1}) v_{\mathrm{b}} + (q_{\mathrm{w}})_{\mathrm{n}} - (q_{\mathrm{w}})_{\mathrm{n}+1} \right]. \qquad (4)$$

The remaining information required to complete all calculations are statements for water and ice pressures, temperature and a criterion for lens initiation.

Darcy's law

The flux of water through the fringe is assumed to obey Darcy's law

$$\frac{du_{\mathrm{w}}}{dz} = f_{\mathrm{w}} - \frac{q_{\mathrm{w}}}{k_{\mathrm{w}}}. \qquad (5)$$

This relationship introduces the soil function k_{w}, the hydraulic conductivity. Each soil will have its unique hydraulic conductivity function. It is not a specific property of heaving soils, but is rather a general material property of all frozen soils, heaving or non-heaving. It must be known in order to calculate the heaving process.

To date, most efforts to calculate this material property are by inference. It is inferred from thermal analysis (van Loon et al. 1988), back calculations (Ratkje et al. 1982) and unsubstantiated inference to non-frozen soil (Guymon and Luthin 1974). Black and Miller (1990) were able to directly measure the change in hydraulic conductivity in air-free, lens-free and solute-free frozen soil as a function of unfrozen water content. Their analysis found that if the measured hydraulic conductivity was expressed a function of the difference between the ice and water pressures

$$\phi = u_{\mathrm{i}} - u_{\mathrm{w}} \qquad (6)$$

then the analogous expression given by Brooks and Corey (1964) for partially saturated and ice-free soil could be transformed to

$$W(\phi) = (W_{\mathrm{sat}} - W_{\mathrm{d}}) \left(\frac{\phi_{\mathrm{b}}}{\phi} \right)^{\alpha} + W_{\mathrm{d}} \qquad (7)$$

and

$$k_{\mathrm{w}}(\phi) = (k_{\mathrm{w}})_{\mathrm{sat}} \left(\frac{\phi_{\mathrm{b}}}{\phi} \right)^{\beta}. \qquad (8)$$

Fourier's law

The flux of thermal energy through the fringe is assumed to follow the Fourier's law

$$\frac{d\theta}{dz} = -\frac{q_{\mathrm{h}}}{k_{\mathrm{h}}}. \qquad (9)$$

This relationship introduces another soil function, k_{h}, the thermal conductivity for the frozen soil. Just as the hydraulic conductivity, this material property is also a function of the pressure difference between the ice and water pressures. One standard expression is the geometric mean formulation of Farouki (1981)

$$k_{\mathrm{h}}(\phi) = (k_h)_{\mathrm{G}}^{e} (k_{\mathrm{h}})_{\mathrm{W}}^{W(\phi)} (k_{\mathrm{h}})_{\mathrm{I}}^{I(\phi)}. \qquad (10)$$

Clapeyron equation

The equilibrium condition between the ice and water pressures and temperature is given by the Clapeyron equation

$$\frac{d\phi}{dz} = (Y-1)\frac{du_w}{dz} - \frac{Yh}{\theta_o}\frac{d\theta}{dz}. \tag{11}$$

Criterion for lens initiation

To calculate when a new lens forms and the lensing cycle is complete, a criterion for lens initiation is required. Simply stated, a new lens will form when ice can penetrate between the grains, causing the grains to separate. Other factors must be included if the porous material was not granular. For example, cements and rocks that have chemical bonds between the grains must have the bonds broken before the ice can penetrate.

Geotechnical engineers express the state of stress within a granular material by the Terzaghi equation

$$\sigma_T = \sigma_e + \sigma_n \tag{12}$$

and the condition when the grains separate by

$$\sigma_e = 0. \tag{13}$$

At this instance

$$\sigma_T = \sigma_n. \tag{14}$$

Now if σ_T is assumed to be equal to the maximum heaving pressure, then the location where a new ice lens forms is where σ_n also equals the maximum heaving pressure. Snyder and Miller (1985) found that they could correctly model their empirically measured neutral stress data by

$$\sigma_n = \chi u_w + (1-\chi)u_i \tag{15}$$

in which the new soil function was determined to be

$$\chi(\phi) = \frac{1}{2}\left[\frac{W(\phi)-W_d}{W_{sat}-W_d} - \frac{0.3}{\phi_n}\sum_{j=1}^{n}\frac{W(\phi_j)-W_d}{W_{sat}-W_d}\right]. \tag{16}$$

Algorithm

The solution strategy begins by stating values for v_i, v_b and $(d\theta/dz)_b$ and all necessary param-

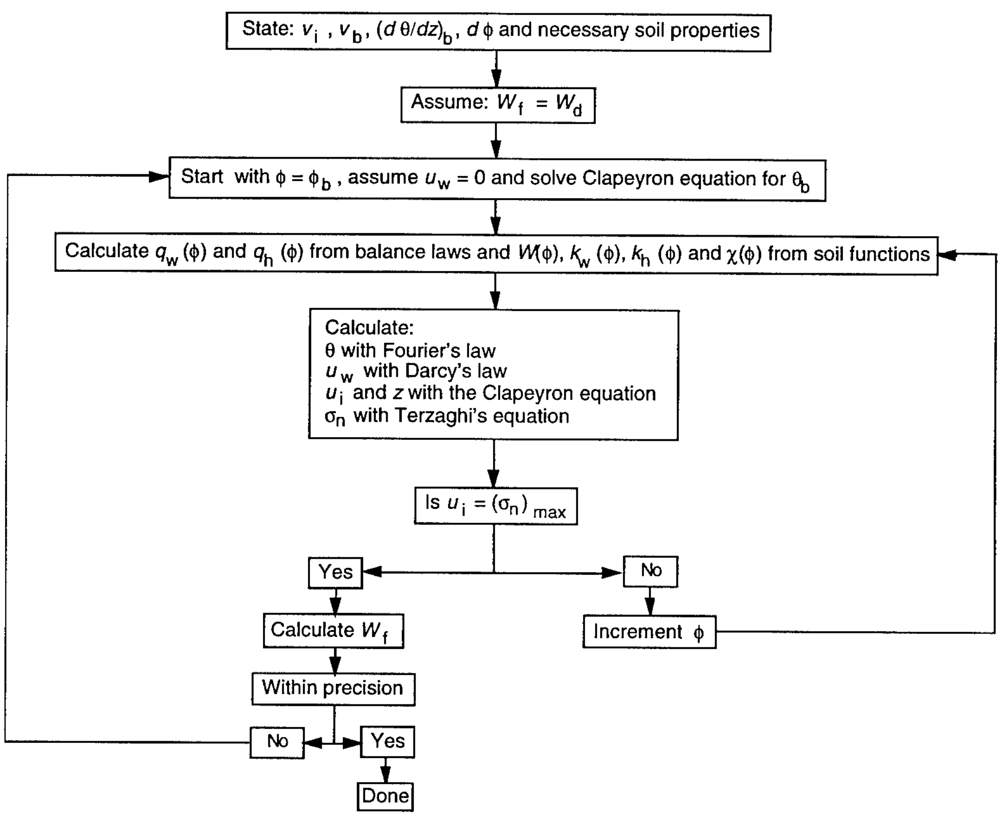


Figure 2. Flow chart of RIGIDICE.

eters for the soil functions. Next, the residual unfrozen water content in the mature frozen soil is assumed to be equal to the lower limit of freezing. Equations 1 and 2 are then solved for the water flux into the fringe and the flux of heat out of the upper boundary.

Profiles of pressures, stresses and temperature are then calculated within the fringe starting at the freezing front. The value of ϕ_b at the freezing front is obtained by setting the local water pressure to zero. This is equivalent to assuming that the water table is at the freezing front. If the water pressure is not zero, then the resulting heaving pressure obtained by assuming zero must be adjusted using eq 11.

Calculations are done in terms of ϕ. In finite difference form, balances are conducted across layers of constant $d\phi$. This has the benefit of generating thin spatial layers where ϕ is changing rapidly and large spatial layers where it changes least.

Darcy's law gives the value of the water pressure across the layer and Fourier's law gives the temperature. The Clapeyron equation then gives the ice pressure as well as the thickness of the layer. Neutral stress is calculated for each layer and a running account of its magnitude recorded to determine its maximum value. A running sum of the unfrozen water content is also made, starting from the location of maximum neutral stress to determine the residual water content. Calculations continue until the ice pressure equals the maximum neutral stress. This is the location of the base of the lens and the new lens will form where the neutral stress was maximum. The distance between these two locations gives the lens spacing. The calculated residual water content is then compared to the original guess. If the difference is unacceptable, the layer by layer calculations are performed again with the new guess until the resulting change in residual water content is acceptable.

A flow chart representation of this algorithm is presented in Figure 2.

RIGIDICE WITH OBJECT-ORIENTED NUMERICS (OON)

Code was written to solve eq 1 through 16 using the algorithm outlined above. To allow for the greatest flexibility of use, as well as ease of future enhancements, C++ was used. The entire code is presented in Appendix A.

The benefit of using OON is that it allows the writing of code in separate layers that are easily merged together. This might be interpreted as the standard approach of writing a series of subroutines, but it is more. The traditional approach is to write a series of procedures to numerically solve the problem. These procedures are sequentially solved and in large programs result in many pages of critically linked lines of code. When, for example, the routine to calculate hydraulic conductivity must be changed, that section of code must be found and modified. Care must be taken not to remove variables that are used by the rest of the program as well as not to add variables that are already being used in other parts of the program.

The OON approach is to break the problem down into separate self-contained units called classes, a class being a collection of data and operations. Data in one class can be made inaccessible to other classes to prevent inadvertent changes by later modifications to the program. Additional classes can be made that inherit the properties of existing classes. Operations used by a class can be expanded by a process called polymorphism. Again, if a new function to calculate hydraulic conductivity is needed, it is not necessary to write a completely new class, but just merely to add to the current. In other words, the new function would access all the data used by the old as well as any new data that it requires, and the new data are prevented from interfering with any existing data in the program. The traditional approach leads to mistakes and debugging problems. The OON approach rests on the belief that if the original code worked, then don't change it, just add improvements.

RIGIDICE is setup as a series of C++ classes. At the lowest level, functions and data associated with a particular soil are grouped together in a class called *Tsoil*. Functions and data associated with the boundary conditions are grouped together as a class *Tbnds* and numerical precision in a class *Ttol*. Since all variables are reduced to dimensionless form to help in numerical calculations, a class that does scaling is called *REDUCE*. Finally, the calculations that perform the necessary algorithm are made in the class *Trigidice*. This class is declared so that it inherits all the properties of the other classes. If a different algorithm is necessary, then a new one can be written that also inherits the other classes while never touching the original source code for the other primitive classes.

In total, the program requires the initiation of 18 variables. It then returns the values of eight calculated variables. Figure 3 shows the input screen of the MathCad program that runs RIGIDICE.

This program calculates the rate of heave for a given set of initial and boundary conditions. The strategy is the same as presented by Black and Miller (1985). It is assumed that the heaving process can be approximated with a time step called the lensing cycle and that any fluctuations during this lensing cycle average out to values that represent the true behavior during the lensing cycle.

Scaling factors

$\lambda := 1.0 \cdot 10^{-6}$

$\zeta := 1.0 \cdot 10^{-2}$

$fw := 1$

Thermal conductivities

$Khw := 0.52$

$Khi := 2.32$

$Khg := 3.42$

Soil water properties

$Wsat := 0.42$

$Wd := 0.02$

$Kwsat := 1.0 \cdot 10^{-8}$

Brooks and Corey functions

$\phi b := 11.196$

$\alpha := 0.36$

$\beta := 2.6$

Boundary conditions

$heave := 10$

$penetration := 100$

$grad\theta unfrozen := -10$

Numeric settings

$prec := 0.1$

$resol := 0.01$

$MaxLayers := 100000$

$$settings(heave, penetration) := \begin{bmatrix} \lambda \\ \zeta \\ fw \\ Khw \\ Khi \\ Khg \\ Wsat \\ Wd \\ Kwsat \\ \alpha \\ \beta \\ \phi b \\ heave \\ penetration \\ grad\theta unfrozen \\ prec \\ resol \\ MaxLayers \end{bmatrix}$$

The function RIGIDICE takes the contents of settings and returns a 1-dimensional array containing (heave pressure, temperature gradient in frozen soil, heat fluxes into the fringe and out of the upper boundary zone, the water flux into the fringe and the lens spacing and thickness and iterations)

$$(ui \;\; grad\theta f \;\; qhin \;\; qhout \;\; qwin \;\; spacing \;\; thickness \;\; iters) := RIGIDICE(settings[heave, penetration])$$

$ui = 76.06$ $spacing = 0.118$ $thickness = 4.251$ $qhin = 15.504$ $qhout = 199.533$ $iters = 2$ $grad\theta f = -70.255$

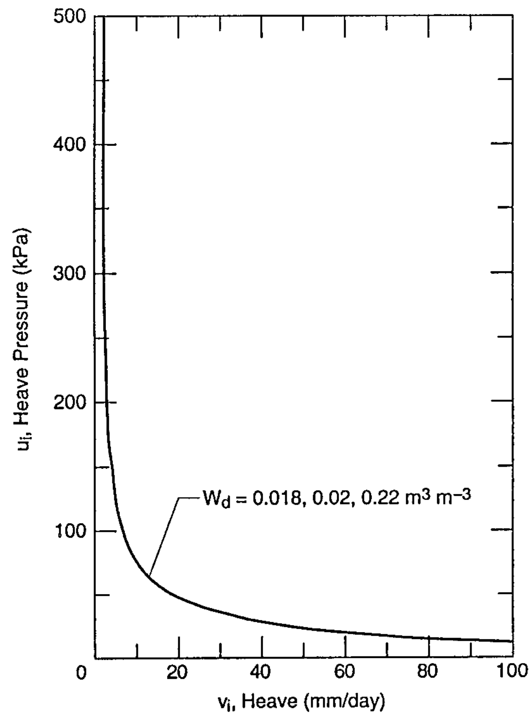
Figure 3. MathCad program to run RIGIDICE.

The user is free to modify the contents of all 18 input parameters that are passed to the array *settings*. The function *RIGIDICE (settings)* returns an array that contains the eight calculated variables.

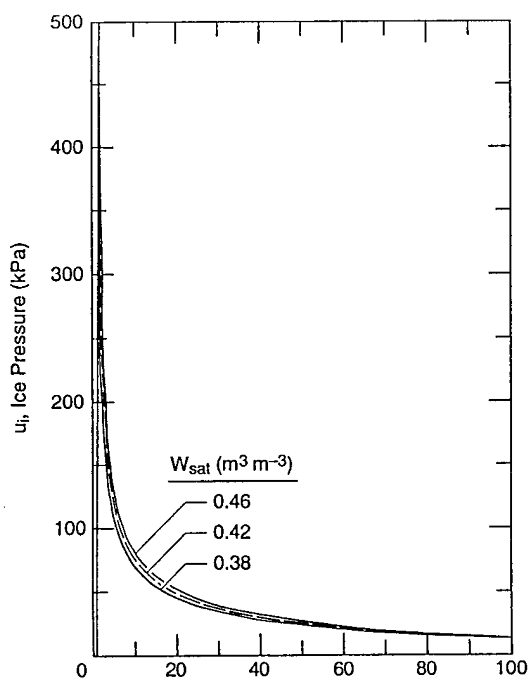
DISCUSSION

Owing to the number of input parameters, a thorough sensitivity study of the program is very time consuming. This study employed a quicker approach in which the relative significance of the various input parameters is obtained through a simple scheme of sequentially varying one parameter at a time and noting the resulting effect on the important calculated parameters. Each input parameter is modified by 10% and the calculated results noted. The initial reference value for each parameter is obtained from past measurements for one particular silty soil (Black and Miller 1985, 1990).

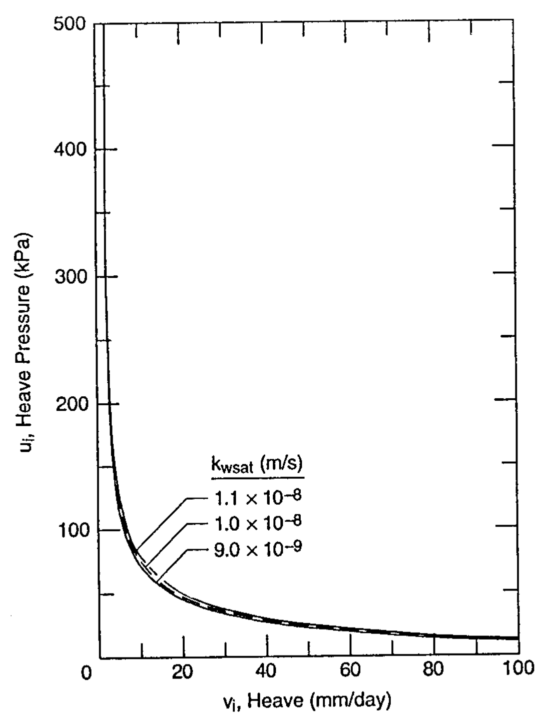
It is evident that the heaving pressure is of primary concern to the end user. The heaving pressure is therefore the output parameter to be examined. In addition to examining how the heave pressure alone changes with each parameter, the heave rate and heave pressure behavior will be examined as the other parameters are modified one at a time. This approach is similar to how an end user might use the program. By stating all the other parameters, the end user is then able to predict heave rate as a function of heave pressure (i.e., overburden pressure).



b. Lower limit of freezing, W_d.

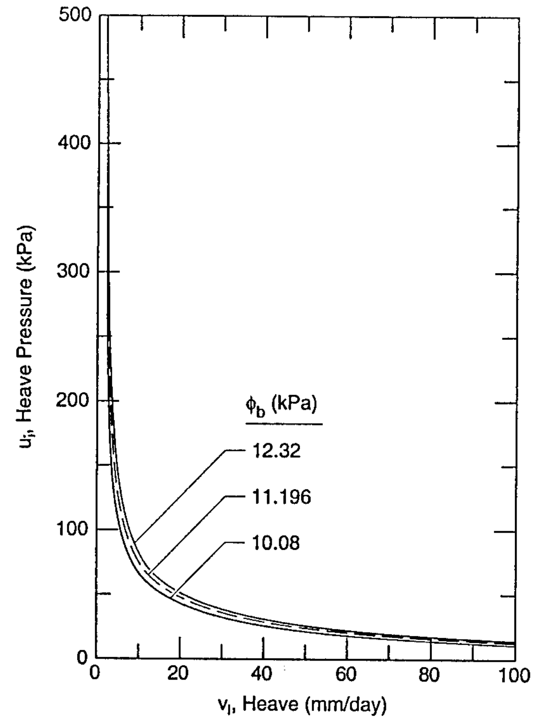


a. Saturated water content, W_{sat}.

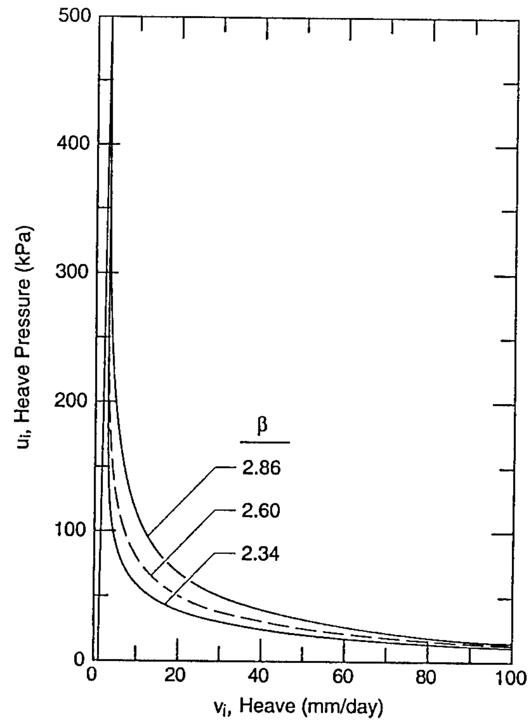


c. Saturated hydraulic conductivity, $(k_w)_{sat}$.

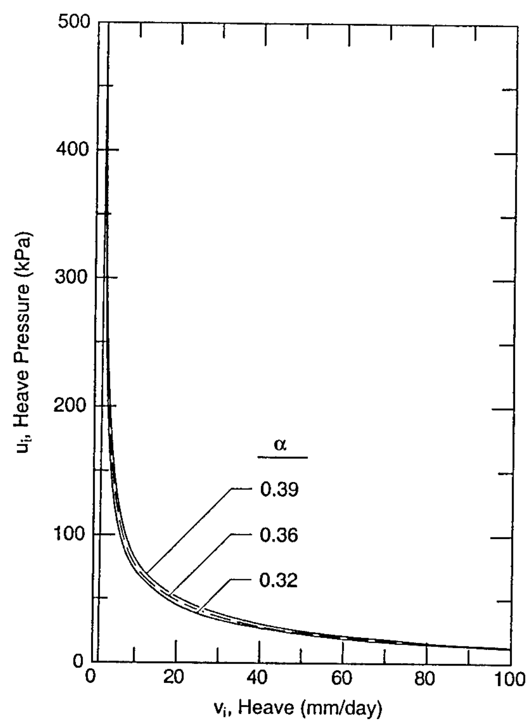
Figure 4. Changes in heave pressure vs. heave rate for changes in chosen input parameters.



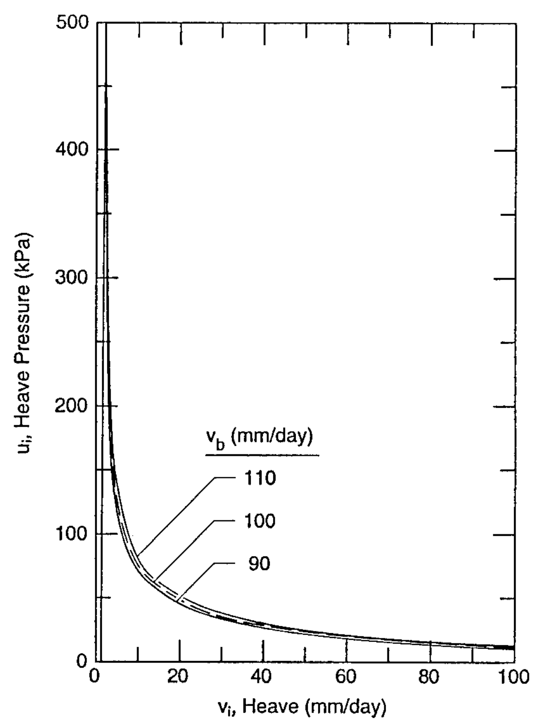
d. Ice entry pressure, ϕ_b.



f. β in Brooks and Corey equation.



e. α in Brooks and Corey equations.

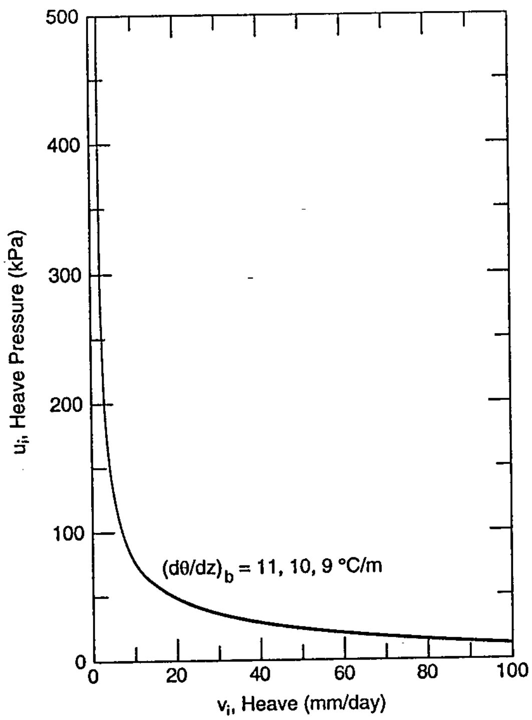


g. Penetration rate, v_b.

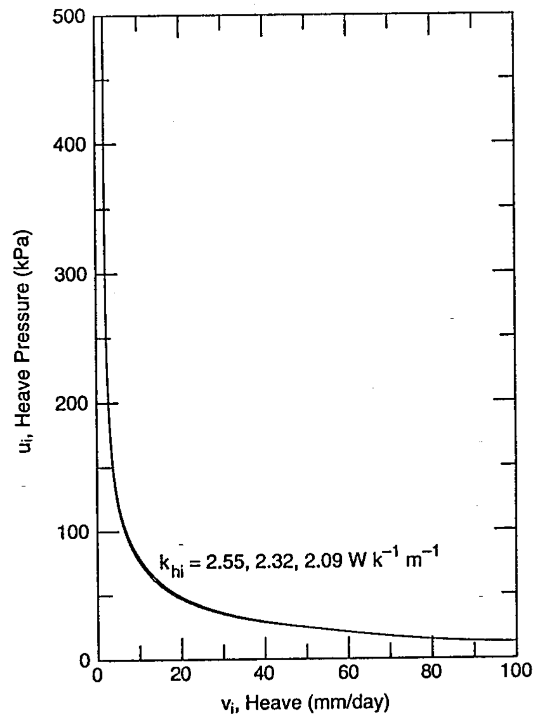
Figure 4 (cont'd). Changes in heave pressure vs. heave rate for changes in chosen input parameters.

The strategy is to examine the influence that changes in the 12 chosen input parameters have on the magnitude of the predicted heaving pressure. This is accomplished by running simulations that vary the value of these parameters by 10% from the initial reference values in Table 1. The resulting predicted heaving pressures are then computed and displayed in Table 1 for direct comparison along with a series of graphs that show the overall effect on the heave pressure as a function of heave rate.

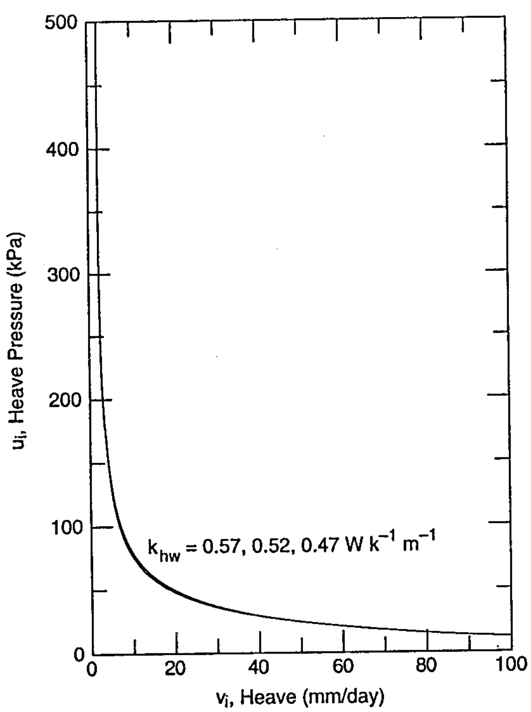
This approach is important in designing experimental tests to aid in the development of engi-



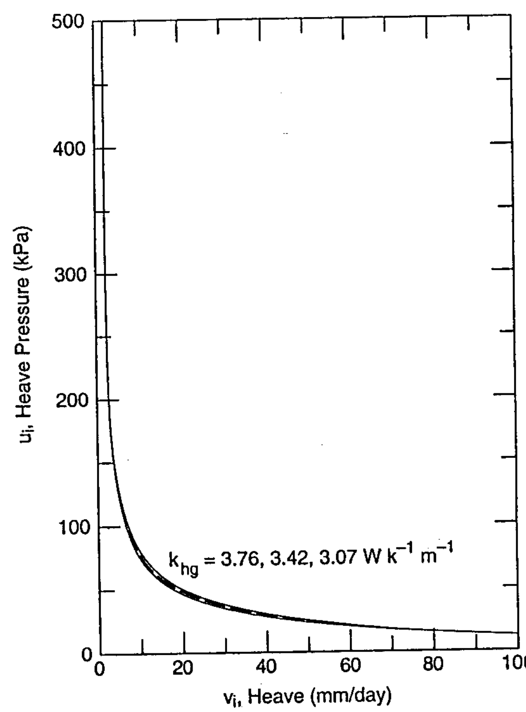
h. Temperature gradient in the unfrozen soil, $\nabla\theta$.



j. Thermal conductivity of ice, $(k_h)_i$.



i. Thermal conductivity of water, $(k_h)_w$.



k. Thermal conductivity of soil grains, $(k_h)_g$.

Figure 4 (cont'd).

neering design criteria. Variables that cause the greatest change in predicted heaving pressure are assumed to be important. The end user should therefore concentrate on obtaining correct values for these important variables if the predicted results are to be useful.

Figures 4a, b and c show how a 10% change in either direction from the reference value for the three soil water parameters influences the calculated heave pressure for a given heave rate. In either case, the resulting deviation is less than the 10% change in the control parameter. A 10% change

Table 1. Sensitivity results of calculated heave pressure to input parameter changes in RIGIDICE.

		Calculated heave pressure (kPa)		
Program parameter	*Reference value*	*0.9 Reference relative error**	*Reference*	*1.1 Reference relative error**
Soil water properties				
W_{sat}	0.42 ($m^3\,m^{-3}$)	70.22 −7.7	76.06	82.05 7.9
W_d	0.02 ($m^3\,m^{-3}$)	76.08 2.6	76.06	75.72 0.4
$(k_w)_{sat}$	1×10^{-8} ($m\,s^{-1}$)	72.33 −4.9	76.06	79.58 4.6
Brooks and Corey constants				
ϕ_b	11.196 (kPa)	67.94 −10.7	76.06	81.82 7.6
α	0.36	71.55 −5.9	76.06	79.86 5.0
β	2.6	109.03 43.3	76.06	56.86 −25.2
Boundary conditions				
v_i	10 ($mm\,day^{-1}$)	81.38 7.0	76.06	71.39 −6.1
v_b	100 ($mm\,day^{-1}$)	71.59 −5.9	76.06	80.24 5.5
$\nabla\theta$	−10 ($^{\circ}C\,m^{-1}$)	75.47 −0.8	76.06	76.38 0.4
Thermal conductivities				
$(k_h)_w$	0.52 ($W/K^{-1}\,m^{-1}$)	76.38 0.4	76.06	75.38 −0.9
$(k_h)_i$	2.32 ($W/K^{-1}\,m^{-1}$)	76.74 0.9	76.06	75.12 −1.2
$(k_h)_g$	3.42 ($W/K^{-1}\,m^{-1}$)	77.86 2.4	76.06	74.33 −2.3
Scaling factors				
λ	1.0×10^{-6} (m)			
ζ	1.0×10^{-2} (m)			
f_w	1 ($m\,s^{-2}$)			
Numeric settings				
Precision	0.1			
Resolution	0.01			
Maximum layers	1.0×10^{5}			

$$^{*}\textit{relative error}\ (\%) = \frac{\textit{value} - \textit{reference}}{\textit{reference}}\,100$$

in the saturated water content W_{sat} resulted in the largest variation of the three, as listed in Table 1, with a relative error range of –8 to 8%. These results suggest that a 10% uncertainty in the magnitude of W_{sat}, W_d and $(k_w)_{sat}$ should not have a severe adverse effect on the predicted heave pressure.

The largest relative errors result from a 10% change in the Brooks and Corey constants ϕ_b and β for the soil as shown in Figures 4d and 4f. Table 1 lists the largest range of 43 to –25% relative error in the calculated heave pressure for a 10% change in β and a –10 to 8% error for ϕ_b. Clearly, this indicates that the hydrologic properties of the soil must be known if any accurate heave behavior is to be calculated. This is unfortunate since this information is rarely even collected. Figure 4e and Table 1 shows a smaller response, similar to the soil water properties, for a change in α.

Surprisingly, a 10% change in the boundary conditions also gives a similar response in relative error to heave pressure as the soil water properties. Figures 4g and h show the response for a change in penetration v_b and the temperature gradient in the unfrozen soil $\nabla\theta$. The heave rate v_i response to calculated heave pressure u_i is obtained from any of the graphs as it is always the center reference line. Again, this is unfortunate since the penetration rate and temperature gradient are easily obtained from temperature profiles. This means that more accurate temperature measurements will not necessarily result in better predictive capability.

The last group of graphs displays the response to an uncertainty in thermal conductivities. Inspection of Figures 4i, j and k along with Table 1 reveals that an uncertainty in this group of parameters has the least influence of all parameters on the calculated heave pressure.

CONCLUSIONS

The past problem of all frost heave models was the trade-off between physical correctness and ease of use. Those models that are easy to calculate tend to be based upon curve fitting heave experiments (Blanchard and Fremond 1985) or incorrectly applying phase equilibrium (Harlan 1973). The models that are physically based are difficult to implement and require time-consuming computation (O'Neill and Miller 1985). Past efforts to overcome this dilemma relied on modifying the original equations of Miller by simplifying the physics to make the mathematics trivial (Gilpin 1980, Holden 1983) or attempting to maintain the physics while leading to a reasonable solution strategy (Black and Miller 1985). This later approach was successfully employed in this paper.

The differential equations of secondary frost heave (Miller 1978) are numerically solved in finite difference form with the program RIGIDICE. By choosing the language C++ and its object oriented nature, the core program is easily attached to the mathematical analysis program MathCad 5.0+. The ease of analysis in this new format allows simulations to be conducted in any physically correct model that the end user requires.

A brief sensitivity analysis of several important parameters shows that current practices of ground freezing monitoring are flawed. Calculated behavior for heave rate and pressure was found to be largely insensitive to penetration rate and temperature gradients. Likewise, unfrozen water content uncertainty did not strongly influence the heave rate and pressure behavior. Hydraulic conductivity, though, was found to have a dramatic influence.

These results indicate that the effort expended in making accurate temperature profiles in freezing ground is not necessary. It also indicates that the great efforts required to monitor unfrozen water content changes in the field might also be unnecessary. The sensitivity study did demonstrate the importance of the hydraulic conductivity. A wise use of research and development efforts should therefore be to develop techniques to measure the hydraulic conductivity in the laboratory and monitor it in the field.

The current model relies upon assumed standard expressions for thermal, hydraulic and stress behavior in soil. Other expressions need to be explored (i.e., van Genuchten's expression for hydraulic properties) in order to generalize the applicability of this model. The OON approach employed will make this a reasonable task. Empirical testing of this model is currently under way with a refrigerated centrifuge. This approach will test the scaling laws for freezing (Miller 1990) as well as the predictions of this model.

LITERATURE CITED

Black, P.B. and R.D. Miller (1985) A continuum approach to modeling frost heaving. *Freezing and Thawing of Soil–Water Systems, Technical Council on Cold Regions Engineering Monograph* (D.M. Ander-

son and P.J. Williams, Ed.). American Society of Civil Engineers, p. 36–45.

Black, P.B. and R.D. Miller (1990) Hydraulic conductivity and unfrozen water content of airfree frozen silt. *Water Resources Research,* **26**(2): 323–329.

Blanchard, D. and M. Fremond (1985) Soil frost heaving and thaw settlement. *Fourth International Symposium on Ground Freezing, 5–7 August, Sapporo, Japan.*

Brooks, R.H. and A.T. Corey (1964) Hydraulic properties of porous media. Hydrology Paper no. 3, Civil Engineering Department, Colorado State University, Fort Collins.

Farouki, O.T. (1981) Thermal properties of soil. USA Cold Regions Research and Engineering Laboratory, Monograph 81-1.

Gilpin, R.R. (1980) A model for prediction of ice lensing and frost heave in soils. *Water Resources Research,* **16**: 918–930.

Guymon, G.L. and J.N. Luthin (1974) A coupled heat and moisture transport model for arctic soils. *Water Resources Research,* **10**(5): 995–1001.

Harlan, R.L. (1973) Analysis of coupled heat-fluid transport in partially frozen soil. *Water Resources Research,* **9**: 1314–1322.

Holden, J.T. (1983) Approximate solutions for Miller's theory of secondary heave. In *Proceedings, 4th International Conference on Permafrost, July 17–22, Fairbanks, Alaska.* Washington, D.C.: National Academy Press, p. 373.

MathSoft (1994) MathCad Plus 5.0. Cambridge, Massachusetts.

Miller, R.D. (1978) Frost heaving in non-colloidal soils. In *Proceedings, 3rd International Conference on Permafrost, 10–13 July, Edmonton, Alberta.* Vol. 1. Ottawa: National Research Council of Canada, p. 708–713.

Miller, R.D. (1990) Scaling in soil physics: principles and applications. Soil Science Society of America Special Publication 25.

O'Neill, K. and R.D. Miller (1985) Exploration of a rigid ice model of frost heave. *Water Resources Research,* **21**(3): 281–296.

Ratkje, S.K., H. Yamamoto, T. Takashi, T. Ohrai and J. Okamoto (1982) The hydraulic conductivity of soils during frost heave. *Frost I Jord,* **24**: 22–26.

Snyder, V.A. and R.D. Miller (1985) Tensile strength of unsaturated soils. *Soil Science Society of America Journal,* **49**: 58–65.

van Loon, W.K.P., I.A. van Haneghem and H.P.A. Boshoven (1988) Thermal and hydraulic conductivity of unsaturated frozen sands. In *Proceedings of the 5th International Symposium on Ground Freezing, Balkema, Rotterdam,* p. 81–90.

Wong, M.K.W., K.G. Budge, J.S. Peery and A.C. Robinson (1993) Object-oriented numerics: a paradigm for numerical object-oriented programming. *Computers in Physics,* **7**(6): 655–663.

APPENDIX A: RIGIDICE MODEL

RIGIDICE.H

```
// acceleration due to gravity
#define grav ((const double) 9.800)      //   m/sec**2
// density of liquid water
#define rhoW ((const double) 1.000E03)   //   kg/m**3
// density of ice
#define rhoI ((const double) 0.917E03)   //   kg/m**3
// viscosity of water
#define nu ((const double) 1.787E-3)   //   kg/m*s
// specific surface free energy of a ice-air interface
0  #define gammaIA ((const double) 0.100)      //   n/m

// Latent heat of fusion per unit volume of melt
#define hIW ((const double) 3.335E05)    //   j/m**3
// standard melting point of bulk ice, or its analogue, when exposed
// to air at standard atmospheric pressure
#define theta0 ((const double) 273.15)     //   kelvins
// conversions
// mm/day / (m/s)
#define mmpdTomps ((const double) 8.64e7)
// kPa to Pa
#define kP2P ((const double) 1.00e3)


class REDUCE{
protected:
            double      orglamda,
                        orgeta,
                        orgFW;
public:
// Constructor
            REDUCE();
            void InitScales();
            void FirstScales(double micro, double macro, double body);
            double      lamda;
            double      eta;
//          double      body;
            double FW;
            double Y;
            double L(double rSI);
            double P(double pSI) ;
// reduced latent heat of fusion, per unit volume of melt
            double H;
            double RTHETA(double thetaSI);
            double SIGMA(double sigmaSI);
            double GRAD(double gradSI);
            double F(double fSI);
            double KW(double kwSI);
            double V(double vSI);
```

```
            double DIV(double divSI);
            double T(double tSI);
            double C(double cSI);
            double KH(double khSI);
            double QH(double qhSI);
};


class Tsoil{
public:

            REDUCE R;
// saturated volumetric water content
            double WSAT;
// lower limit of drying
            double WD;
// reduced saturated hydraulic conductivity
            double KWSAT;

            public:
// Constructor
        Tsoil();
            void InitWATER(double wsat, double wd, double ksat);
// calculates water content from degree of saturation
            double W (double DSAT);

              private:
// volume fraction of ice
            double I;
// volume fraction of soil
            double G;
// thermal conductivity of water
            double KHW;
// thermal conductivity of ice
            double KHI;
// thermal conductivity of soil grains
            double KHG;
public:
            void InitTherm(double khw, double khi, double khg);
// calculates thermal conductivity for given composition
            double RKH (double W);
// returns the protected reduced ice conductivity
            double RKHI();

protected:
```

```
// unfrozen water content exponent
            double ALPHA;
private:
// frozen hydraulic conductivity exponent
            double BETA;
protected:
// reduced ice intrusion pressure
            double PHIB;
public:
            void InitBC(double alpha, double beta, double phib);
            double DSAT(double PHI);
            double RKW(double PHI);

// running sum
private:
            double PROD;
            int FlagSny;
public:
// initialize starting parameters
            void InitStress();
// calculates Snyder's factor
            double SNYDER (double PHI, double DSAT, double deltaDSAT);
};


// boundary conditions
class Tbnds
            {
            public: REDUCE R;
            protected:
                        // heave rate
            double VI, VIorg,
                        // penetration rate
                                    VB, VBorg,
                        // unfrozen temperature gradient
                                    GRADTB, GRADTBorg,
                        // sought after heave pressure
                                    UIorg;

            public:
            void InitQuasi(double press, double pene, double gradtb);
            void InitContinuum(double heave, double pene, double gradtb);
            Tbnds();
            };


// variables used for numeric precision
class Ttol
            {
            protected:
            double      orgresolution,
                        orgprecision;

         public:
```

```
                    double precision,     // check on Wd
                    resolution,           // layer thickness
                    LCMAX;                // cutoff number of layers

                    double DPHI;
                    void InitTol();
                    void FirstTol(double prec, double resol, double lcmax);
                    Ttol();
          };


// variables used in calculations
class Trigidice:
          public Tbnds,
          public Ttol,
          public REDUCE,
          public Tsoil
          {
          public:
          double    QW1, QW2,// reduced mass flux
                              DW,// reduced change in water content across layer
                              GRADUW1, GRADUW2,// reduced water pressure gradient
                              QH1, QH2,// reduced thermal flux
                              GRADTHETA1, GRADTHETA2,// reduced thermal gradient
                              PHI,// reduced phi
                              GRADPHI1, GRADPHI2,// reduced gradient of PHI from Clapeyron
                              Z,// reduced thickness
                              ZN,// reduce location of new lens
                              ZI,// reduce location of old lens
                              DZ,// reduced layer thickness
                              UW,// reduced water pressure
                              DUW,// reduced change in water pressure across layer
                              UI,// reduced ice pressure
                              CHI,//reduced stress partition factor
                              UN,// reduced neutral stress
                              UNold,// reduced previous neutral stress
                              UNmax,// reduced maximum neutral stress
                              THETA,// reduced temperature
                              DTHETA,// reduced temperature change across layer
                              WF,// unfrozen water in mature frozen zone
                              FSI;

          Trigidice();
          void LAYER(double rphi, double rdphi);
          void PROFILE();
          void GLOBAL();

          double oldPHI;
          double sum;
          int iteration,
          // error signal for GRADPHI < 0
                    GRADPROB,
          // not enough layers
```

```
                     LAYERPROB;
        double tic,
        // number of layers
                     counter;
        double maybe;
        int check();
        int prec();
        void errors();
        void once();
        int pressdiff();
        int slope();
        void quasi();
        void InitTCALC();
        double heave();
        double iterout();
        double pressure();
        double tgradout();
        double heatin();
        double heatout();
        double waterin();
        double spacing();
        double thickness();
        void look();
};
```

RIGIDICE.CPP

```
#include <iostream.h>
#include <math.h>
#include "rigidice.h"

//          REDUCED VARIABLES
// From:
// R. D. Miller (1990) Scaling of freezing phenomena,
// in Scaling in soil physics:principles and applications
// SSSA special Publication Number 25
0

            REDUCE::REDUCE() {
// specific gravity of ice?
            Y=rhoI/rhoW;
            orglamda=1.0e-6;
            orgeta=0.01;
            FW = 1;
            orgFW = F(FW*(-rhoW*grav));
// reduced latent heat of fusion, per unit volume of melt
            H =rhoW*(lamda/gammaIA)*hIW;
            InitScales();
            }

void REDUCE::FirstScales(double micro, double macro, double body){
            orglamda = micro;
            orgeta = macro;
            FW = body;
            orgFW = F(FW*(-rhoW*grav));
// reduced latent heat of fusion, per unit volume of melt
            H =rhoW*(lamda/gammaIA)*hIW;
            InitScales();
}

void REDUCE::InitScales(){
            lamda = orglamda;
            eta = orgeta;
            FW = orgFW;
            }

// reduced length
            double REDUCE::L(double rSI)
                        { return ((1/lamda)*rSI);}
// reduced pressure
            double REDUCE::P(double pSI)
                        { return ((lamda/gammaIA)*pSI);}
// reduced temperature
            double REDUCE::RTHETA(double thetaSI)
                        { return ((1/theta0)*thetaSI);}
// reduced stress
            double REDUCE::SIGMA(double sigmaSI)
                        { return ((lamda/gammaIA)*sigmaSI);}
// reduced gradient
```

```
            double REDUCE::GRAD(double gradSI)
                        { return (eta*gradSI);}
// reduced body force
            double REDUCE::F(double fSI)
                        { return ((lamda*eta/gammaIA)*fSI);}
// reduced capillary conductivity
            double REDUCE::KW(double kwSI)
                        { return ((nu/(lamda*lamda))*kwSI);}
// reduced velocity
            double REDUCE::V(double vSI)
                        { return ((eta*nu/(lamda*gammaIA))*vSI);}
// reduced divergence
            double REDUCE::DIV(double divSI)
                        { return (eta*divSI);}
// reduced time
            double REDUCE::T(double tSI)
                        { return ((lamda*gammaIA/(eta*eta*nu))*tSI);}
// reduced volumetric heat capacity
            double REDUCE::C(double cSI)
                        { return ((lamda*theta0/gammaIA)*cSI);}
// reduce thermal conductivity
            double REDUCE::KH(double khSI)
                        { return ((nu*theta0/(gammaIA*gammaIA))*khSI);}
// reduced heat flux
            double REDUCE::QH(double qhSI)
                        { return (eta*nu/(gammaIA*gammaIA)*qhSI);}

//
//
//
//                      SOIL FUNCTIONS
//

Tsoil::Tsoil(){
            InitWATER(0.42,0.02,1.0e-8);
            InitBC(0.36,2.6,11.196);
            InitTherm(0.52,2.32,3.42);
            InitStress();
            }

void Tsoil::InitWATER(double wsat, double wd, double ksat){
            // saturated volumetric water content
//          WSAT = 0.42;
            WSAT = wsat;
```

```
            // lower limit of drying
//          WD = 0.02;
            WD = wd;
            // saturated hydraulic conductivity
//          KWSAT=1.0e-8;         // m/s
            KWSAT = ksat;
            KWSAT=KWSAT/(rhoW*grav);
            KWSAT = R.KW(KWSAT);
            }

// given degree of saturation
double Tsoil::W(double DSAT){
            return( DSAT*(WSAT-WD)+WD);
                        }
//
//
//          Thermal conductivity

void Tsoil::InitTherm(double khw, double khi, double khg){
// volume fraction of soil
            G=1-WSAT;
// volume fraction of ice
            I=1-G-WD;
// thermal conductivity of water
//          KHW =0.52; // W/m
            KHW = khw;
            KHW = R.KH(KHW);
// thermal conductivity of ice
//          KHI =2.32; // W/m
            KHI = khi;
            KHI = R.KH(KHI);
// thermal conductivity of soil grains
//          KHG=3.42;  // W/m
            KHG = khg;
            KHG = R.KH(KHG);
}

// calculates thermal conductivity for given composition
double Tsoil::RKH (double W){
            I = 1 - G - W;
            return ( pow(KHG,G)*pow(KHW,W)*pow(KHI,I));
            }

// to get KHI
double Tsoil::RKHI(){
            return (KHI);
            }
//

//
//          Brooks & Corey equations
//
void Tsoil::InitBC(double alpha, double beta, double phib) {
//            ALPHA = 0.36;
```

```
//            BETA = 2.6;
              ALPHA = alpha;
              BETA = beta;

             // ice entry pressure
//          PHIB = 11.196*kP2P;   // kPa
              PHIB = phib;
              PHIB = R.P(PHIB*kP2P);
                                        }

// returns degree of saturation for a given phi
double Tsoil::DSAT(double PHI) {
                        return (pow((PHIB/PHI),ALPHA));
                        }

// hydraulic conductivity for a give phi
double Tsoil::RKW(double PHI) {
             //         return (bc.KWSAT*pow((PHIB/PHI),BETA));
            return (KWSAT*pow((PHIB/PHI),BETA));}


//
//          Stress partition factor
//
//

            void Tsoil::InitStress(){
// set defaults to obtain a Snyder factor of 1
            PROD=0.0;
            FlagSny = 0;
}
// calculates Snyder's factor
            double Tsoil::SNYDER (double PHI,
                                                  double DSAT,
                                           double deltaDSAT){
                        if (FlagSny == 0) {
                                  FlagSny = 1;
                                  PROD = 0;
                                  return (1);
                                  }
                                                   else{
                        PROD = PROD + PHI*deltaDSAT;
                        return( 0.5* (DSAT-(0.3/PHI)*PROD));
                        }
                                                   }


Tbnds::Tbnds(){
              InitQuasi(100,10,-10);
              InitContinuum(10,100,-10);
            }


void Tbnds::InitContinuum(double heave, double pene, double gradtb){
```

```
            VI = heave;
            VIorg = R.V(VI/mmpdTomps);
            VI = VIorg;
            VB = pene;
            VBorg = R.V(VB/mmpdTomps);
            VB = VBorg;
            GRADTB = gradtb;
            GRADTBorg = R.GRAD(R.RTHETA(gradtb));
            GRADTB = GRADTBorg;
}

void Tbnds::InitQuasi(double press, double pene, double gradtb){

            UIorg = R.P(press*kP2P);
            VBorg = R.V(pene/mmpdTomps);
            GRADTBorg = R.GRAD(R.RTHETA(gradtb));
            }

Ttol::Ttol(){
            FirstTol(0.1,0.1,100);
            }

void Ttol::FirstTol(double prec, double resol, double lcmax){
            orgprecision = prec;
            orgresolution = resol;
            LCMAX= lcmax;
            InitTol();
            }

void Ttol::InitTol(){
            precision = orgprecision;
            resolution = orgresolution;
            }


Trigidice::Trigidice(){
            // specific gravity of ice?
//          Y=rhoI/rhoW;
//          lamda=1.0e-6;
//          eta=0.01;
//          FW = 1.0;
//          FW = F(FW*(-rhoW*grav));
// reduced latent heat of fusion, per unit volume of melt
//          H =rhoW*(lamda/gammaIA)*hIW;
            InitTCALC();
            }

void Trigidice::InitTCALC(){
            QW1=0;
            QW2=0;// reduced mass flux
            DW=0; // change in water content across layer
            GRADUW1=0;
            GRADUW2=0;// reduced water pressure gradient
            QH1=0;
```

```
            QH2=0;// reduced thermal flux
            GRADTHETA1=0;
            GRADTHETA2=0;// reduced thermal gradient
            GRADPHI1=0;
            GRADPHI2=0;// reduced gradient of PHI from Clapeyron
            Z=0;// reduced thickness
            ZN=0;//reduced location of new lens
            ZI=0;//reduced location of old lens
            DZ=0;// reduced layer thickness
            UW=0;//reduced water pressure
            DUW=0;// reduced change in water pressure
                                    // across layer
            UI=PHI;// reduced ice pressure
            CHI=0;//reduced value of the stress partition factor
            UN=0;// reduced neutral stress
            UNold=0;// previous value of reduced neutral stress
            UNmax=-1000;// maximum value of neutral stress
            THETA=0;
            DTHETA=0;
            GRADPROB=0;
            LAYERPROB=0;
            InitStress();
            InitTol();
            }


void Trigidice::LAYER(double rphi, double rdphi){
            DW = W(DSAT(rphi))-W(DSAT(rphi+rdphi));
            QW2 = QW1 + (VB-Y*(VB+VI))*DW;
            GRADUW2 = FW - QW2/RKW(rphi+rdphi);
            QH2 = QH1 + H*(DW*VB + (QW2 - QW1));
            GRADTHETA2 = -QH2/RKH(W(DSAT(rphi+rdphi)));
            GRADPHI2 = (Y-1)* GRADUW2 - Y*H* GRADTHETA2;
            if ((GRADPHI1<0)||(GRADPHI2< 0)) {
                        GRADPROB=1;
                        DZ=1;
            } else
            DZ = rdphi/(sqrt(GRADPHI1*GRADPHI2));
            DUW = FW*DZ - DZ*(QW1/RKW(rphi) +
                        QW2/(RKW(rphi+rdphi)))/2;
            DTHETA = ((Y-1)*DUW-rdphi)/(Y*H);
// reset variables for beginning of next layer
            QW1=QW2;
            GRADUW1=GRADUW2;
            QH1=QH2;
            GRADTHETA1=GRADTHETA2;
            GRADPHI1=GRADPHI2;
            }

void Trigidice::PROFILE(){
            LAYER(PHI,DPHI);
            PHI=PHI+DPHI;
            DPHI = resolution/DZ;
            THETA=THETA+DTHETA;
```

```
            Z=Z+DZ;
            UW=UW+DUW;
            UI=PHI+UW;
            CHI=SNYDER(PHI,DSAT(PHI),DSAT(PHI+DPHI)-DSAT(PHI));
            UN=CHI*UW+(1-CHI)*UI;
            }

void Trigidice::GLOBAL(){
            InitTCALC();
            QW1 = Y*VI + ((Y-1)*(WSAT-WF))*VB;
            GRADUW1 = FW - QW1/KWSAT;
            QH1 = -RKH(WSAT)*GRADTB ;
            FSI = VI/(VI+VB);
            GRADTHETA1 = GRADTB;

            GRADPHI1 = (Y-1)* GRADUW1 - Y*H* GRADTHETA1;
            DPHI = resolution*PHIB*sqrt(GRADPHI1);
            DZ = DPHI/GRADPHI1;
            resolution = DZ*DPHI;
            UW=0;
            UI=PHI;
            CHI=SNYDER(PHI,DSAT(PHI),DSAT(PHI+DPHI)-DSAT(PHI));
            UN=CHI*UW+(1-CHI)*UI;
            THETA=-PHI/(Y*H);
            }



int Trigidice::prec(){

            if (( fabs((sum/(ZI-ZN)-WF)/WD) <= precision)
                        || (LAYERPROB) || (GRADPROB)){
            return 0;
                          }
            else{
                        WF=sum/(ZI-ZN);
                                                            iteration++;
                        return 1;
            }
            }


int Trigidice::check(){

            if (counter > LCMAX){
                        LAYERPROB=1;
                        errors();
                        iteration = -999;
                        return 0;
                        }
            if (GRADPROB){
                        errors();
                        iteration = -888;
                        return 0;
```

```
                    }
          if (UN > UNmax){
                    UNmax=UN;
                    ZN=Z;
                    return 1;
                                                    }
          if ((UN <= UNmax)&& (UI>UNmax)){
                    sum=sum+DW*DZ;
                    ZI=Z;
                    return 1;
                                                    }

          if ((UI<=UNmax) && (counter>=2)){
                    return 0;
                    }
                    return 1;
          }


void Trigidice::errors(){
                    WF=0;
                    UI=0;
                    ZN=0;
                    ZI=1;
                    VI=0;
                    }


void Trigidice::once(){
                    WF=WD;
                    PHI = PHIB;
                    iteration = 1;
                    do{
                              counter = 0;
                              PHI = PHIB;
                              sum = 0;
                              GLOBAL();
                              do {
                                        counter++;
                                        PROFILE();
                                        } while ( check());
                      } while ( prec());
          }



int Trigidice::pressdiff(){


          if ( fabs(UIorg-UI) <= P(precision))
                    {
                    return 0;
                    }
          else{
```

```
                        VI=VI+VIorg;
                                                }

            if (/*(LAYERPROB)||*/(UI<UIorg))
                                    {
                                    VI = VI - 2.0*VIorg;
                                    VIorg=VIorg/2.0;
                                    VI = VI + VIorg;
//                                  LAYERPROB = 0;
                                    }
            if (VI<0.0)
                        {
                        VI = V(0.5/mmpdTomps);
                                                            }

            return 1;

            }

int Trigidice::slope(){
            if(GRADPROB){
                        orgresolution=orgresolution/2;
                        LCMAX=LCMAX*2;
                        return 1;
                        }
            else{
                        return 0;
                        }
                          }


void Trigidice::quasi(){

                        VB = VBorg;
                        GRADTB = GRADTBorg;
                        VI = V(0.5/mmpdTomps);
                        VIorg = V(10/mmpdTomps);
                        resolution=orgresolution;
                        do{
                                    do{
                                                once();
                                                }while(slope());
                        } while (pressdiff());
            }

double Trigidice::heave(){

            quasi();
            return (VI * gammaIA*lamda/(eta*nu)* mmpdTomps);
            }

double Trigidice::pressure(){
            once();
            return ((UI*gammaIA/lamda)/kP2P);
```

```
            }

double Trigidice::iterout(){
            return(iteration);
            }

double Trigidice::tgradout(){
            QH2 = -RKH(WSAT)*GRADTB+H*(VB*(WSAT-WF)+ Y*VI + ((Y-1)*(WSAT-WF))*VB);
            return(-1*theta0/eta*QH2*(FSI/RKHI()+(1-FSI)/RKH(WF)));
            }
double Trigidice::heatin(){
            return(-RKH(WSAT)*GRADTB/QH(1));
            }
double Trigidice::heatout(){
            QH2 = -RKH(WSAT)*GRADTB+H*(VB*(WSAT-WF)+ Y*VI + ((Y-1)*(WSAT-WF))*VB);
            return(QH2/QH(1));
            }

double Trigidice::waterin(){
            QW1 = Y*VI + ((Y-1)*(WSAT-WF))*VB;
            return(QW1*gammaIA*lamda/(eta*nu));
            }

double Trigidice::spacing(){
            return((ZI-ZN)*VI/VB * eta *1000);
            }

double Trigidice::thickness(){
             return(ZI * eta *1000);
             }

void Trigidice::look(){
                        cout <<
                        iteration << ","
            << counter  << ","
            << FSI << ","
            << (UI*gammaIA/lamda)/kP2P<< ","
            << VI * gammaIA*lamda/(eta*nu)* mmpdTomps << ","
            << VB * gammaIA*lamda/(eta*nu)* mmpdTomps << ","
            << GRADTB*theta0/eta << ",";
            QW1 = Y*VI + ((Y-1)*(WSAT-WF))*VB;
            QH1 = -RKH(WSAT)*GRADTB ;
            QH2 = QH1+H*(VB*(WSAT-WF)+QW1);
//          fout << -1.0*theta0/eta*QH2*(VI*RKH(WF)+VB*RKH(0))/((VI+VB)*(RKH(0)*RKH(WF)))
            cout <<-1.0*theta0/eta*QH2*(FSI/RKHI()+(1-FSI)/RKH(WF))
                                    << ","
            << QW1*gammaIA*lamda/(eta*nu) << ","
            << (ZI-ZN)*VI/VB * eta *1000 << ","
            << ZI * eta *1000
            << "\n";
            }
```

HEAVE.CPP

```
#include "mcadincl.h"
#include "rigidice.h"


        LRESULT RIGIDICEFunction(      LPCOMPLEXARRAY      calculated,
                                                LPCCOMPLEXARRAY settings);


        FUNCTIONINFO    RIGIDICE =
        {
        "RIGIDICE",                     // Name by which mathcad will recognize the function
        "settings",                  // heaverate will be called as heavepressure(settings)
       "pressure, heat and mass fluxes and lens sizes for given settings",            //
       description of heavepressurre(settings)
        (LPCFUNCTION)RIGIDICEFunction,      // pointer to the executible code
        COMPLEX_ARRAY,                      // the return type is a complex scalar
        1,                                   // the function takes on 1 argument
        { COMPLEX_ARRAY}                   // that is an array
        };


        LRESULT RIGIDICEFunction(      LPCOMPLEXARRAY      calculated,
                                                LPCCOMPLEXARRAY settings)
        {

              Trigidice     foo;
              double transient;
              double   micro, macro, body;
              double   khw, khi, khg;
              double   wsat, wd, ksat;
              double   alpha, beta, phib;
              double   prec, resol, lcmax;
              double   heave, pene, gradtb;

              micro = settings->hReal[0][0];
              macro = settings->hReal[0][1];
              body = settings->hReal[0][2];
              foo.FirstScales( micro, macro, body);

              khw = settings->hReal[0][3];
              khi = settings->hReal[0][4];
              khg = settings->hReal[0][5];
              foo.InitTherm( khw, khi, khg);

              wsat = settings->hReal[0][6];
              wd = settings->hReal[0][7];
              ksat = settings->hReal[0][8];
              foo.InitWATER( wsat, wd, ksat);

              alpha = settings->hReal[0][9];
              beta = settings->hReal[0][10];
```

```
        phib = settings->hReal[0][11];
        foo.InitBC( alpha, beta, phib);

        heave = settings->hReal[0][12];
        pene = settings->hReal[0][13];
        gradtb = settings->hReal[0][14];
        foo.InitContinuum( heave, pene, gradtb);

        prec = settings->hReal[0][15];
        resol = settings->hReal[0][16];
        lcmax = settings->hReal[0][17];
        foo.FirstTol( prec, resol, lcmax);

        MathcadArrayAllocate( calculated, 1, 8, TRUE, FALSE);

        transient = foo.pressure();
        calculated->hReal [0] [0] = transient;
        transient = foo.tgradout();
        calculated->hReal [0] [1] = transient;
        transient = foo.heatin();
        calculated->hReal [0] [2] = transient;
        transient = foo.heatout();
        calculated->hReal [0] [3] = transient;
        transient = foo.waterin();
        calculated->hReal [0] [4] = transient;
        transient = foo.spacing();
        calculated->hReal [0] [5] = transient;
        transient = foo.thickness();
        calculated->hReal [0] [6] = transient;
        transient = foo.iterout();
        calculated->hReal [0] [7] = transient;



        return 0;              // return 0 to indicate there was no error

    }


    LRESULT heavepressureFunction(     LPCOMPLEXSCALAR    pressure,
                                      LPCCOMPLEXARRAY settings);


    FUNCTIONINFO    heavepressure =
    {
    "heavepressure",             // Name by which mathcad will recognize the function
    "settings",              // heaverate will be called as heavepressure(settings)
    "heave pressure for given settings",   // description of heavepressurre(settings)
    (LPCFUNCTION)heavepressureFunction,    // pointer to the executible code
    COMPLEX_SCALAR,                   // the return type is a complex scalar
    1,                                // the function takes on 1 argument
    { COMPLEX_ARRAY}                // that is an array
    };

```

```
LRESULT heavepressureFunction(      LPCOMPLEXSCALAR     pressure,
                                    LPCCOMPLEXARRAY settings)
{

      Trigidice    foo;
      double HeavePressure;
      double   micro, macro, body;
      double   khw, khi, khg;
      double   wsat, wd, ksat;
      double   alpha, beta, phib;
      double   prec, resol, lcmax;
      double   heave, pene, gradtb;

      micro = settings->hReal[0][0];
      macro = settings->hReal[0][1];
      body = settings->hReal[0][2];
      foo.FirstScales( micro, macro, body);

      khw = settings->hReal[0][3];
      khi = settings->hReal[0][4];
      khg = settings->hReal[0][5];
      foo.InitTherm( khw, khi, khg);

      wsat = settings->hReal[0][6];
      wd = settings->hReal[0][7];
      ksat = settings->hReal[0][8];
      foo.InitWATER( wsat, wd, ksat);

      alpha = settings->hReal[0][9];
      beta = settings->hReal[0][10];
      phib = settings->hReal[0][11];
      foo.InitBC( alpha, beta, phib);

      heave = settings->hReal[0][12];
      pene = settings->hReal[0][13];
      gradtb = settings->hReal[0][14];
      foo.InitContinuum( heave, pene, gradtb);

      prec = settings->hReal[0][15];
      resol = settings->hReal[0][16];
      lcmax = settings->hReal[0][17];
      foo.FirstTol( prec, resol, lcmax);

      HeavePressure = foo.pressure();
      pressure->real = HeavePressure;


      return 0;              // return 0 to indicate there was no error

}

LRESULT heaverateFunction(       LPCOMPLEXSCALAR     heaverate,
                                 LPCCOMPLEXARRAY settings);
```

```
FUNCTIONINFO    heaverate =
{
"heaverate",                 // Name by which mathcad will recognize the function
"settings",                 // heaverate will be called as heavepressure(settings)
"heave rate for given settings",     // description of heavepressurre(settings)
(LPCFUNCTION)heavepressureFunction,     // pointer to the executible code
COMPLEX_SCALAR,                     // the return type is a complex scalar
1,                                  // the function takes on 1 argument
{ COMPLEX_ARRAY}                   // that is an array
};


LRESULT heaverateFunction(          LPCOMPLEXSCALAR      heaverate,
                                   LPCCOMPLEXARRAY settings)
{

      Trigidice     foo;
      double HeaveRate;
      double   micro, macro, body;
      double   khw, khi, khg;
      double   wsat, wd, ksat;
      double   alpha, beta, phib;
      double   prec, resol, lcmax;
      double   press, pene, gradtb;

      micro = settings->hReal[0][0];
      macro = settings->hReal[0][1];
      body = settings->hReal[0][2];
      foo.FirstScales( micro, macro, body);

      khw = settings->hReal[0][3];
      khi = settings->hReal[0][4];
      khg = settings->hReal[0][5];
      foo.InitTherm( khw, khi, khg);

      wsat = settings->hReal[0][6];
      wd = settings->hReal[0][7];
      ksat = settings->hReal[0][8];
      foo.InitWATER( wsat, wd, ksat);

      alpha = settings->hReal[0][9];
      beta = settings->hReal[0][10];
      phib = settings->hReal[0][11];
      foo.InitBC( alpha, beta, phib);

      press = settings->hReal[0][12];
      pene = settings->hReal[0][13];
      gradtb = settings->hReal[0][14];
      foo.InitQuasi( press, pene, gradtb);

      prec = settings->hReal[0][15];
      resol = settings->hReal[0][16];
```

```
                lcmax = settings->hReal[0][17];
                foo.FirstTol( prec, resol, lcmax);
                HeaveRate = foo.heave();
                heaverate->real = HeaveRate;
                return 0;               // return 0 to indicate there was no error
        }
BOOL WINAPI DllEntryPoint (HANDLE hDLL, DWORD dwReason, LPVOID lpReserved)
{
  switch (dwReason)
  {
        case DLL_PROCESS_ATTACH:
   {
                // DLL is attaching to the address space of the current process.
                //
                  if ( CreateUserFunction( hDLL, &RIGIDICE ) == NULL )
                        break;
                  if ( CreateUserFunction( hDLL, &heaverate ) == NULL )
                        break;
                  if ( CreateUserFunction( hDLL, &heavepressure ) == NULL )
                        break;
        }
  //            CreateUserFunction( hDLL, &heavepressure );
       case DLL_THREAD_ATTACH:     // A new thread is being created in the current process.
       case DLL_THREAD_DETACH:      // A thread is exiting cleanly.
       case DLL_PROCESS_DETACH:   // The calling process is detaching the DLL from its address space.
           break;
  }
  return TRUE;
}
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1995	

4. TITLE AND SUBTITLE

RIGIDICE Model of Secondary Frost Heave

5. FUNDING NUMBERS

PE: 6.27.84A
PR: 4A762784AT42
TA: BS
WU: CP-C01

6. AUTHORS

Patrick B. Black

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Cold Regions Research and Engineering Laboratory
72 Lyme Road
Hanover, New Hampshire 03755-1290

8. PERFORMING ORGANIZATION REPORT NUMBER

CRREL Report 95-12

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of the Chief of Engineers
Washington, D.C. 20314-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

Available from NTIS, Springfield, Virginia 22161

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A revised version of an earlier attempt to numerically solve Miller's equations for the RIGIDICE model of frost heave is presented that corrects earlier mistakes and incorporates recent improvements in the scaling factors of ground freezing. The new version of the computer code also follows the concepts of Object Oriented Numerics (OON), which allow for easy modification and enhancements. Analysis of the program is accomplished with the symbolic math program MathCad. A brief sensitivity analysis of the input variables indicates that those parameters that calculate the hydraulic conductivity have the greatest influence on the variability of predicted heaving pressure.

14. SUBJECT TERMS

Frost heave	Hydraulic conductivity	Thermal conductivity
Frozen soil	Stress	

15. NUMBER OF PAGES 41

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL